PR #44648 完整报告

vllm-project/vllm

[Bugfix] [ROCm] [Critical] fallback to regular abi for ROCm

合并时间：2026-06-05 23:51

原文链接：http://prhub.com.cn/vllm-project/vllm/pull/44648

执行摘要

此 PR 修复了 ROCm 因 PyTorch 2.10 不支持稳定 ABI 导致的构建失败。通过条件编译，将 get_cuda_view_from_cpu_tensor 等算子从稳定 ABI 回退到非稳定 ABI，仅影响 ROCm 路径，CUDA 不受影响。

功能与动机

PR #44334 将 silu_and_mul_per_block_quant 移入稳定 ABI，但稳定 ABI 需要 PyTorch 2.11 版本。ROCm 官方当前仅支持 PyTorch 2.10，导致编译错误（见 issue #44641）。本 PR 通过回退相关算子到非稳定 ABI 来解决，并预留 TODO 以待 ROCm 版本升级后清理。

实现拆解

1. 非稳定 ABI 注入：在 csrc/torch_bindings.cpp 的 TORCH_LIBRARY_EXPAND 中添加 get_cuda_view_from_cpu_tensor 和 _cuda_utils 库的注册，由 #ifdef USE_ROCM 保护。
2. 稳定 ABI 条件编译：在 csrc/libtorch_stable/torch_bindings.cpp 中将相关注册移到 #ifndef USE_ROCM 下，ROCm 不编译。
3. 新增核心实现：创建 csrc/cuda_view.cu 包含旧的完整 UVA 实现，仅在 ROCm 构建时编译。
4. 头文件声明调整：更新 csrc/ops.h 和 csrc/libtorch_stable/ops.h，确保声明与编译路径匹配。
5. 构建系统适配：CMakeLists.txt 设置 ROCm 下 TORCH_TARGET_VERSION=2.10，并调整源文件列表。

csrc/torch_bindings.cpp

非稳定 ABI 入口，添加了 ROCm 下 get_cuda_view_from_cpu_tensor 和 _cuda_utils 的注册，是回退的核心。

```
// csrc/torch_bindings.cpp（修改后版本的关键添加部分）
#include "cuda_utils.h"

TORCH_LIBRARY_EXPAND(TORCH_EXTENSION_NAME, ops) {
  // ... 先前已有 op 注册 ...

  // ---- ROCm 回退：因 ROCm 使用 torch 2.10，稳定 ABI 无法编译 ----
  #ifdef USE_ROCM
```

```
  // TODO: 移除此块当 ROCm 升级到 torch 2.11
  ops.def("get_cuda_view_from_cpu_tensor(Tensor cpu_tensor) -> Tensor");
  ops.impl("get_cuda_view_from_cpu_tensor", torch::kCPU,
           &get_cuda_view_from_cpu_tensor);
  #endif

  // ... 后续其他 op 注册 ...
}

// 同样在文件末尾添加 _cuda_utils 库注册
// ( 对于 ROCm，这些不能在稳定 ABI 中编译，故放在这里 )
#ifdef USE_ROCM
TORCH_LIBRARY_EXPAND(CONCAT(TORCH_EXTENSION_NAME, _cuda_utils), cuda_utils) {
  cuda_utils.def("get_device_attribute(int attribute, int device_id) -> int");
  cuda_utils.impl("get_device_attribute", &get_device_attribute);
  cuda_utils.def(
    "get_max_shared_memory_per_block_device_attribute(int device_id) -> int");
  cuda_utils.impl("get_max_shared_memory_per_block_device_attribute",
                  &get_max_shared_memory_per_block_device_attribute);
}
#endif
```

csrc/libtorch_stable/torch_bindings.cpp

稳定 ABI 库的注册文件，将 get_cuda_view_from_cpu_tensor 和 _cuda_utils 的注册移入 #ifndef USE_ROCM 保护，确保 ROCm 不编译。

```
// csrc/libtorch_stable/torch_bindings.cpp ( 修改部分 )
// 在 stable ABI 库中，ROCm 跳过这部分注册

STABLE_TORCH_LIBRARY_IMPL(_C, CUDA, ops) {
  // ... 原有大量 op 实现 ...
}

// ----- ROCm 回退：稳定 ABI 部分仅对 CUDA 生效 -----
#ifndef USE_ROCM
// TODO: 移除此保护块当 ROCm 升级到 torch 2.11
STABLE_TORCH_LIBRARY_IMPL(_C, CPU, ops) {
  ops.impl("get_cuda_view_from_cpu_tensor",
           TORCH_BOX(&get_cuda_view_from_cpu_tensor));
}

// 注册 _cuda_utils 稳定 ABI 库
STABLE_TORCH_LIBRARY_FRAGMENT(_C_cuda_utils, cuda_utils) {
  cuda_utils.def("get_device_attribute(int attribute, int device_id) -> int");
  cuda_utils.def(
    "get_max_shared_memory_per_block_device_attribute(int device_id) -> int");
}

STABLE_TORCH_LIBRARY_IMPL(_C_cuda_utils, CompositeExplicitAutograd, cuda_utils) {
```

```
  cuda_utils.impl("get_device_attribute", TORCH_BOX(&get_device_attribute));
  cuda_utils.impl("get_max_shared_memory_per_block_device_attribute",
                  TORCH_BOX(&get_max_shared_memory_per_block_device_attribute));
}
#endif
```

csrc/cuda_view.cu

新增文件，包含 get_cuda_view_from_cpu_tensor 在非稳定 ABI 下的完整实现，是 ROCm 回退的运行时核心。

```
// csrc/cuda_view.cu ( 新增文件，ROCm 专用，待 torch 升级后移除 )
// TODO: Remove this once ROCm upgrade to torch 2.11.

#include <torch/all.h>
#include <torch/cuda.h>
#include <cuda_runtime.h>

torch::Tensor get_cuda_view_from_cpu_tensor(torch::Tensor& cpu_tensor) {
  // 确保输入是 CPU 张量
  TORCH_CHECK(cpu_tensor.device().is_cpu(), "Input tensor must be on CPU");

  // 处理空张量
  if (cpu_tensor.numel() == 0) {
    return torch::empty(cpu_tensor.sizes(),
                        cpu_tensor.options().device(torch::kCUDA));
  }

  if (cpu_tensor.is_pinned()) {
    // 如果 CPU 张量是 page-locked 的，直接获取设备指针
    void* host_ptr = const_cast<void*>(cpu_tensor.data_ptr());
    void* device_ptr = nullptr;
    cudaError_t err = cudaHostGetDevicePointer(&device_ptr, host_ptr, 0);
    TORCH_CHECK(err == cudaSuccess,
                "cudaHostGetDevicePointer failed: ", cudaGetErrorString(err));

    // 构建一个 CUDA 视图张量，引用同一个数据
    return torch::from_blob(
        device_ptr, cpu_tensor.sizes(), cpu_tensor.strides(),
        [base = cpu_tensor](void*) {}, // 保持 CPU 张量存活
        cpu_tensor.options().device(torch::kCUDA));
  }

  // 如果 CPU 张量未 pin，分配新的 pinned 缓冲区并拷贝数据
  torch::Tensor contiguous_cpu = cpu_tensor.contiguous();
  size_t nbytes = contiguous_cpu.nbytes();

  void* host_ptr = nullptr;
  cudaError_t err = cudaHostAlloc(&host_ptr, nbytes, cudaHostAllocMapped);
  if (err != cudaSuccess) {
```

```
    AT_ERROR("cudaHostAlloc failed: ", cudaGetErrorString(err));
  }

  err = cudaMemcpy(host_ptr, contiguous_cpu.data_ptr(), nbytes,
                   cudaMemcpyDefault);
  if (err != cudaSuccess) {
    cudaFreeHost(host_ptr);
    AT_ERROR("cudaMemcpy failed: ", cudaGetErrorString(err));
  }

  void* device_ptr = nullptr;
  err = cudaHostGetDevicePointer(&device_ptr, host_ptr, 0);
  if (err != cudaSuccess) {
    cudaFreeHost(host_ptr);
    AT_ERROR("cudaHostGetDevicePointer failed: ", cudaGetErrorString(err));
  }

  // 使用 lambda 在张量析构时释放 pinned 内存
  auto deleter = [host_ptr](void*) { cudaFreeHost(host_ptr); };

  return torch::from_blob(device_ptr, contiguous_cpu.sizes(),
                          contiguous_cpu.strides(), deleter,
                          contiguous_cpu.options().device(torch::kCUDA));
}
```

评论区精华

- 作者在评论中解释了文件移动的意图：将同一条件编译块分组在一起。
- 作者说明 cuda_view.cu 是旧实现，待 ROCm 升级后移除。
- 审核者 Harry-Chen 批准，无争议。

风险与影响

当前风险较低，但需要关注：

- ROCm 路径使用非稳定 ABI，若 PyTorch 2.10 未来发生接口不兼容，可能需要紧急适配。
- 所有 TODO 需要跟踪，待 ROCm 升级 torch 后清理。

影响：ROCm 用户恢复构建，CUDA 用户不受影响。这是关键修复，使 vLLM 在 ROCm 上可用。

关联脉络

本 PR 是 PR #44334（稳定 ABI 迁移）的直接后续修复，与 issue #44641 绑定。未来 ROCm 的 torch 升级后，需通过反向条件编译移除所有回退代码，恢复统一稳定 ABI。